FIT3077 - Sprint 3

Alex Ung, Romal Patel, Hirun Hettigoda

Table of Contents

Code Review

Assessment Criteria

Completeness of solution - [functional completeness as well as correctness]

Is the rationale appropriate? [functional appropriateness]

Understandability of the solution [appropriateness recognisability]

Extensibility [modifiability]

Quality of written source code (coding standards, reliance on case analysis and/or down-casts) [maintainability]

Aesthetics (how good is the UI)

Criteria	0 - Unsatisfactory	1 - Needs Improvement	2 - Satisfactory	3 - Very Good	4 - Excellent
Functional Completeness	Key functionalities are mostly absent or incorrectly implemented.	Some functionalities are present but several key aspects are flawed or missing.	Basic functionalities are present but some are missing or flawed.	Most key functionalities are covered and correctly implemented.	All key functionalities are comprehensively covered and correctly implemented.
Functional Appropriateness	Solution direction lacks clear rationale or is inappropriate for the project goals.	Solution somewhat aligns with project goals but significant adjustments needed.	Solution direction has a basic rationale that aligns somewhat with project goals.	Solution direction is well-reasoned and aligns well with project goals.	Solution direction is excellently reasoned and perfectly aligns with project goals.

Appropriateness Recognisability	Solution direction is unclear or confusing, making it hard to understand.	Solution is somewhat understandable but lacks clarity in important areas.	Solution direction is understandable but could be clearer or more intuitive.	Solution direction is clear and well-understood by stakeholders.	Solution direction is exceptionally clear and easily understood by all stakeholders.
Modifiability	Solution is rigid and difficult to modify for future extensions.	Modification is possible but requires significant effort.	Solution allows for some modification but with considerable effort.	Solution is fairly easy to modify and adapt for future extensions.	Solution is designed with excellent foresight, making it very easy to extend and modify.
Maintainability	Code is poorly written with no adherence to coding standards; heavy reliance on case analysis and down-casts.	Some adherence to standards, but significant issues in code quality.	Code mostly adheres to standards with some reliance on case analysis and down-casts.	Code is well-written with minor issues; minimal reliance on case analysis and down-casts.	Code is exemplary, fully adheres to coding standards and avoids unnecessary complexities.
User Engagement	UI is poorly designed, not engaging, and hard to use.	UI is functional but lacks appeal and minimal engagement.	UI design is functional but lacks visual appeal or user engagement.	UI is visually appealing and engages users effectively.	UI is exceptionally designed, highly engaging, and enhances user experience.

Romal Design Review

Referring to the Assessment Criteria outlined above, Romal's tech-based prototype has been assessed as the following:

Functional Completeness - Very Good (3)

- Romal's implementation demonstrates a complete board setup of the game in which the majority of the functionalities are met, with only a few functionalities not present resulting in a score of 3. The game board has 24 squares/panels around the board in which they are split into their respective volcano cards that are randomised. Specifically going into his chosen functionality the 16 dragon cards are placed in the middle of the board in which they can flip up and down as well as randomise representing a perfect implementation for his chosen section. There are also 4 caves illustrated on the outside of volcano cards which can also be randomised as well as two dragon tokens on the caves which is great. In terms of the board creation and his chosen functionality the functional completeness would result in an excellent score, however taking into account all key functionalities like switch player turn or moving player has not yet been implemented.

Functional Appropriateness - Excellent (4)

- The implementation demonstrates a great start to the game which can be further extended upon. Knowing that creatures are used all around the board (volcano cards, caves, dragon cards) Romal decided to use an abstract Creature class in which other creatures can be extended upon even different types of creatures like PirateDragon. Using his chosen functionality of flipping the dragon card it is illustrated further how going by this approach was a good decision. In the future there may be certain changes to creatures which can be implemented in a respectable manner due to his approach. The caves also take an abstract approach. As of now there may have been a simpler approach since the cave's main attribute is the position, however this approach opens up goals which can be achieved later on in the game. Additionally the game follows all the rules with certain design patterns like singleton also implemented when running the game which is another smart approach as certain factors should have only one instance.

Appropriateness Recognisability: Excellent (4)

- Romal's implementation follows great recognisability. All the code represented is very easy to understand which is further expressed through multiple comments, the structure of the code and naming conventions. Certain files are also placed in specific packages making it easy to navigate. Due to this the solution would be easy to understand from all stakeholders perspectives creating an excellent score.

Modifiability: Excellent (4)

- Since Romal has used abstract classes for the majority of his implementation as well as splitting up certain methods in other classes like flip() in the DragonCard class the code can easily be modified. If the addition of other classes follow the procedure he has taken so far whilst using the abstract classes created it is definitely easily modifiable.

Maintainability: Very Good (3)

- Majority of Romal's implementation is very good in which the code is handled with proper standards and avoids complexities. However for the volcano card creation and placement on the board a function like setupVolcanoCard() takes into account multiple different input parameters which may not be ideal in an implementation and this project.

User Engagement: Excellent (4)

- In terms of Romal's user engagement I would say it's excellent. It is very engaging to the user and the design is great with the way everything is laid out. There is also a great use of images which aligns to the game which is being created. Certain images are also specifically set for certain purposes for example the volcano card images, the image before a dragon card is flipped etc. There are also little details like the brown background colour for the grid the dragon cards are placed in which matches the colour of the image before the dragon card is flipped. The dragon card can also be flipped up and down in a seamless manner representing the image again on the other side when flipped back down which is very good.

Hirun Design Review

Referring to the Assessment Criteria outlined above, Alex's tech-based prototype has been assessed as the following:

1. Functional Completeness:

- Needs Improvement(1): The implementation includes a baseline board setup visually, as there are 24 cards, 4 caves and 16 dragon cards present. And each dragon card position is randomised. The chosen functionality is implemented where each dragon card position is randomised and can be flipped over to reveal a creature. Cave creatures are randomised which is good. However, some improvements need to be made in regards to the ability to unflip the dragon cards, showing creatures on the squares of the volcano cards as well as the randomisation of the board.

2. Functional Appropriateness:

- Very Good (3): The code logically follows the game's rules and objectives as detailed in the game manual. The solution's direction seems appropriate for simulating the board game environment, though some minor improvements could enhance the integration of game components.

3. Appropriateness Recognisability:

- Very Good (3): The source code is structured in a way that it is easy to follow what class does what which makes it easy to understand where Hirun wants to go in terms of his solution direction. His class and method names are descriptive enough to understand their role for the game, making the codebase relatively easy to navigate.

4. Modifiability:

- Very Good (3): The current code in regards to the flipping dragon cards game functionality is easy to modify as the solution is simple with few dependencies which makes it easy to for example add new types of cards or game features. The separation into different classes for handling specific game elements like cards and board setup supports extensibility. There could be some improvements to the modifiability by not hard coding the showing of the board

5. Maintainability:
- Very Good (3): The code is well-organised, with clear separation of responsibilities of each class as well as good separation of responsibilities with the functions related to each class which facilitates maintenance. However, some sections could benefit from additional comments (file headers, function headers) to explain complex logic or decisions.

6. User Engagement (Aesthetics of the User Interface):
- Satisfactory (2): The user interface, based on the description of the `BoardPanel` class, utilises basic Java Swing components. The visuals are most definitely functional, the aesthetic appeal isn't, and the interface could be enhanced to improve user engagement, such as more sophisticated graphics or interactive elements.

Summary:
The code effectively implements the fundamental mechanics of the "Fiery Dragons" game, adhering well to the game's rules and providing a functional user interface. While functionally complete and maintainable, there's room for enhancing the user interface and adding comprehensive documentation to boost understandability and ease of future modifications.

Alex Design Review

Referring to the Assessment Criteria outlined above, Alex's tech-based prototype has been assessed as the following:

<u>Functional Completeness - 3 (Very Good):</u>
In regards to the functional completeness of Alex's tech-based prototype, he has been awarded 'Excellent' for being able to implement close to everything that was required in Sprint 2. By testing and observing the final sprint 2 executable, it can be acknowledged that all cards: Volcano Cards, Dragon Cards, Caves are implemented correctly and work together coherently. Volcano Cards are randomised correctly and the Caves are attached to the correct creature and Volcano Card as well. Furthermore, All Dragon Tokens (players) are added onto the board and also have the ability to move forward and backwards (1-3 steps) seamlessly as per Alex's chosen functionality. The only reason Alex was not given an Excellent is due to the fact that his Dragon Cards were not randomised onto the board. Other than that, his functional completeness is near perfect.

Functional Appropriateness - 4 (Excellent):
Commenting on functional appropriateness, Alex's tech-based prototype is appropriately implemented based on the goal of the game and future extensions. Adding classes such as 'PlayerManager' and 'MovementManager' proves that the prototype efficiently manages the state of the game, handling player turns, movement of tokens and so on. This aligns with the project's goal, and will definitely ease the progression in Sprint 3.

Functional Recognisability - 4 (Excellent):
Observing the code, I can easily understand what each class's and method's purpose is as he has derived clear and informative comments for the reader. Each complex piece of code also has its own comment to outline the goal and purpose of that line making it easier for readers to understand why that line is so important. To add, each variable is carefully named so that it reflects how the variable is being used and why it exists. This makes it easy for readers and future collaborators to work with the pre-existing system. Overall, Alex's code documentation and naming conventions allows other developers to easily and quickly understand the purpose and interconnectedness of the prototype.

Modifiability - 3 (Very Good):
In terms of modifiability, Alex has showcased good practise that allows for future extension and adjustments with minimal disruptions. By implementing a modular design, utilising design patterns such as Singleton for multiple classes (MovementManager, PlayerManager, BoardArray), maintaining clear documentation and creating classes in such a way that there is high cohesion, he has created a framework that supports the introduction of new features, elements and rules to the game. For example, Alex has created specific managers, such as MovementManager and PlayerManager, which gives the class a specific focus and is limited to necessary communication. This is a great example of high cohesion which is a strong factor in creating a highly modifiability codebase and making sure there is not an overarching amount of interdependencies that can cause problems when adding new elements.

Maintainability - 4 (Excellent):
An expectation of a well maintained codebase is one that fully adheres to set out coding standards and avoids unnecessary complexities. Observing the prototype file structure, it can be seen that all classes have been packaged into their relative packages with names that describe the basic nature of these groupings. This is a simple but good practice that promotes maintainability as in the future, as the codebase continues to grow, files can be sorted in a way that is easy to find and manage, increasing efficiency and organisation. To add, as stated before, the prototype consists of great, detailed and straightforward documentation which makes the code easier to maintain as developers can quickly decipher what the code does. Furthermore, Alex's design is simple and straight to the point. Classes and methods are not over complicated and layed out strategically, reducing the chances of errors and enhancing maintainability. Furthermore, overlooking the git, multiple informative commits have been made for each crucial change that has been made to the code which allows for easy backtracking and diagnosing in the arisal of defects.

User Engagement - 1 (Needs Improvement):
Considering that this is the first Sprint that is implementing code, Alex has done a good job in making sure each element of the board is easily distinguishable and identifiable. However, there is a lack in aesthetics that sets the theme of the game as well as graphics to identify the different types of creatures, caves and dragon tokens. Hence, there is a lot to be worked on in terms of a UI that encapsulates the vibrant and dynamic essence of the game. To improve the player experience, the next steps should focus on enhancing the graphical elements that define the visual theme of "Fiery Dragons" and create an immersive environment for our users. Animation is also something to be considered in making the game feel more interactive and appealing if time allows.

Solution Direction:

Alex design was the chosen tech-software prototype for sprint 3. In his design not only did he incorporate his chosen functionality of moving the dragon token but he implemented it in a way where it looks out for the future goals for the game. For example, classes like MovementManager, PlayerManager and Actor provide a great point to add and change code to work in the way we want. Having a separate manager classes also illustrates individual classes with individual responsibilities and methods which results in a clear understanding as well as classes with fewer bundles of code in them. There was also a good layout for the actual board of the game in which UI and other functionalities could be handled well. Overall Alex's code provided a great base for the team to implement on. In terms of the dragon cards and the creatures in the game the team decided to use Romal's code. As a team we thought it would be a better option moving forward if the creatures were represented as separate classes with a main abstract class rather than an enum since it lessens the conditions whilst creating more room for extra functionalities if required. In terms of the dragon cards Romal's implementation also had methods like flip being the functionality he chose which allows the dragon cards to flip up and down. Hirun's code also implemented the dragon cards flipping in a similar fashion with an abstract creature set up for the future however it wasn't directly set up with the dragon cards in Sprint 2 making Romal's implementation a better option. There was also a separate DragonCardPool class in Romal's code which uses a separate method to generate the dragon cards which was better than hard coding the dragon cards within an array. From this the team set out to use Alex's code as the base with Romal's creature and dragon card classes for the game.

Object Oriented Design

CRC Cards

BoardArray	Responsibilities	Collaborators

	Manage singleton instance	VolcanoCard
	Initialises volcano cards	Square
	Shuffles and positions volcano cards	Creature
	Provides access to all the squares across volcano cards	
Description	Manages the board layout for the fiery dragons game, including the creation, shuffling and organisation of volcanoCards which may or may not contain caves.	

Responsibilities	WindowPanel	Collaborators
	Manages the ui for moving panels	BoardArray
	Sets up square panels	SquarePanel
	Sets up caves	CavePanel
	Sets up Players	DragonTokenPanel
	Sets up dragon cards	PlayerManager

		DragonToken
		MovementManager
		Square
		Cave
Description	Manages the graphical display of the game board for the fiery dragons game.	

Responsibilities	DragonToken	Collaborators
	Represents a dragon token in the game	Actor
	Controls what happens on a dragon token turn	Cave
	Tracks the number of squares moved	DragonTokenPanel
	Connects to a DragonTokenPanel	MovementManager
	Owns a cave	
	Represents a dragon token in the game and contains game mechanics related to what the player can do on their turn.	

DragonCard	**Responsibilities**	**Collaborators**
	Display a card that	- Creature

	can be flipped between a front image and a back image	- PlayerManager - MovementManager - DragonToken
	Update the appearance of card when flipped	
	Movement of player based on cards data determined by creature type and creature amount	
	Manage mouse click interactions to flip card	
Manages the dragon card functionalities in the game, in which the dragon token can cause the cards to flip resulting in either the token to move or not based on whether there is a match or not with the creature type. If it is not a match the PlayerManager class is used to switch between players.		

Responsibilities	**MovementManager**	**Collaborators**
	Logic for moving dragon tokens on game board	- dragonToken - BoardArray - WindowPanel - PlayerManager
	Check validity of the moves which determines where a dragon token can	

	legally move to	
	Update dragon token position based on movement	
	Handle wrapping and cave interactions on game board	
Manages the logic for moving dragon tokens on the game board. This class is responsible for checking move validity, updating token positions and managing the interaction between the game rules and user actions		

Responsibilities	**PlayerManager**	**Collaborators**
	Manage player registration and the list of players	- DragonToken - BoardArray - VolcanoCard
	Handle player turns and ensure the correct rotation	
	Add new players and assign dragon tokens to them	
	Reset player count and clear the list of players	
	Assign caves to dragon tokens	

Manages the players within the game, handling player turns, player registration, and assigning caves to the dragon tokens. The class ensures that players are rotated correctly and that each dragon token is assigned to a cave if available

Class Diagram (UML)

Tech-based Software Prototype

Instructions for how to build and run the executable

Video Demonstration

There should be instructions on how to submit this video in the brief/edstem

For the demonstration we'll need to demonstrate these scenarios

- Using a number of game situations, demonstrate that your implementation correctly follows the rules of the game. You must illustrate at a minimum
    - an 'initial' board at the beginning of the game,
    - a player flips a Dragon card,
    - the dragon token moves correctly based on the last uncovered Dragon card,
    - a player uncovers another Dragon card if the turn is not over yet,
    - once a turn is over, flip all the Dragon cards face down for next player's turn,
    - a specific situation where a Dragon Pirate card is uncovered or a different animal (to the animal shown on the square where your dragon stands) is uncovered or your dragon would land on an occupied square, and
    - the end of the game.